A Service Oriented QoS Architecture Targeting the Smart Grid World & Machine Learning Aspects

Christos Chrysoulas and Maria Fasli
School of Computer Science & Electronic Engineering, University of Essex,
Essex, United Kingdom
{cchrys, mfasli}@essex.ac.uk

***Abstract*—Dynamic selection of services and by extension of service providers are vital in today's liberalized market of energy. On the other hand it is equally important for Service Providers to spot the one QoS Module that offers the best QoS level in a given cost. Type of service, response time, throughput, availability and cost, consist a basic set of attributes that should be taken into consideration when building a concrete Grid network. In the proposed QoS architecture Prosumers request services based on the aforementioned set of attributes. The Prosumer requests the service through the QoS Module. It is then the QoS Module that seeks the Service Provider that best fits the needs of the client.**

Keywords— QoS; Service Oriented Architecture; Smart Grid; Mining; Machine Learning

I. INTRODUCTION

In a constantly growing and demanding market of energy environment, there arises the need for a Quality of Service (QoS) mechanism to properly support the constraints that are imposed by the consumers of energy, without neglecting the importance of keeping the balance of energy flow in the network in an as stable as possible level.

In today's liberalized market of energy playground, it is more crucial than ever to seamlessly provide the end users with the requested services, without putting in jeopardy the grid's stability. In order to properly achieve this goal, an in advance way of placing, scheduling, and assigning the requests for energy consumption (or even for energy production) should be considered. A mechanism with respect to attributes like: type of service to be served, response time, availability, cost and probably throughput should be developed and adopted in order to smoothly pass from the classic energy grid to this new more intelligently build Smart Grid era.

In the proposed approach, we try to enforce the Service Oriented Architecture Approach (SOA) to the Smart Grid field. The idea was born by noticing that in the Smart Grid field the whole action is initiated by two main actors, namely the Consumer (in our case the Prosumer/User) and the Provider (in our case the Aggregator) of energy (the service). This is exactly the view from which the SOA is overlooking a system. So we tried to get the best of what this promising SOA field has to offer in order that different Providers be able to independently create their services and seamlessly "feed" the Consumers. This approach is worth adapting to the Smart Grid environment.

The rest of the paper is structured as follows. In Section II, the motivation for bringing QoS in SOAs is described. Section III gives a detailed presentation of the proposed QoS approach. Section IV presents a mining approach for the Smart Grid, while Section V provides the conclusions, and outlines future work.

II. QOS IN SERVICE ORIENTED ARCHITECTURES

SOA is a way of developing software in the form of interoperable services. The promise that the service-oriented development brings to the IT world stems from providing a common programming interface, through which any application can be accessed [1]. A service can be defined as a discrete unit of functionality that is made available through a service contract [2]. The service contract specifies all interactions between the service consumer and service provider and includes: i) Service interface, ii) Interface documents, iii) Service policies, iv) Quality of service (QoS), and v) Performance.

One of the main differences between a service and other software constructs (such as components or objects) is that a service is explicitly managed. The QoS and performance are managed through a service level agreement (SLA). In addition, the entire service life cycle is managed — from design, to deployment, to enhancements, to maintenance.

SOAs can easily support QoS features and behavior by putting their characteristics in the WSDL description of a requested or provided service. Since SOAs message exchange is based on XML, we only need to flourish a bit the description in order to make it possible.

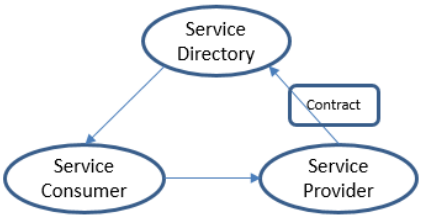

Figure 1 Service Oriented Architecture Overview

Normally the need for code and systems re-use is the driving force for adopting SOAs [3] instead of using highly

specialized building blocks, focusing on a certain application. A service must hide its internal logic. A service should be loosely coupled, with no predefined connections, but with clearly defined inputs and outputs.

QoS in Grid computing was studied in GARA [5]. In GARA approach, the separation of resource reservation and actual allocation is proposed for supporting critical requests. Studies of Ran [6] and Tian [7] concentrated on extending the first one the UDDI registry and the second one extended the WSDL files in order to bridge the gap between the Web Service layer and the network layer. To our knowledge both approaches lack implementation and validation reports.

Numerous approaches for providing QoS support in middleware based models, and specifically message oriented middleware models can be found in the bibliography. The Quartz [8] approach needs a large dataset (meaning large number of attributes) in order to provide adequate QoS support amongst different application areas. In [9] the QoS negotiation is in advance takes place by communicating a QoS contract amongst the involved parties. Our approach is in position to also send alternative offers to the Prosumers.

Cucinota et al [10] presented a SOA approach that allows negotiation of the individuals QoS characteristics. In this way any unwanted interference amongst different services can be avoided. In [11] a negotiation architecture was developed where a QoS Manager detects any possible QoS violations, communicates with the resource manager and starts a new negotiation among the interested parts. Our model is proposing the most fitted to the Prosumer's needs QoS offer based on mining techniques and by processing the outcome with the help of machine learning algorithms.

III. Proposed Architecture

The QoS architecture presented in the paper consists of the following components: the Aggregator [4], the Aggregator Agent (AA), the Prosumer/User [4], the Flex-Offer Agent (FOA) [4], the QoS Agent, the Aggregator Registration, and databases: to store information regarding the Prosumers/Users, the Contracts (closed, served, etc.), and information regarding the available Aggregators and their characteristics. See Figure 2.

The Prosumers/Users send their micro flex-offers to the Aggregator, through the FOA and QoS Module. A micro flex-offer states the possibility of a Prosumer/User to consume a certain amount of energy and the time interval during which it has the flexibility to schedule that consumption. There is also the possibility the flex-offer to be generated by the Flex-Offer Agent or by a Flex-Offer Agent that resides on the Aggregator's side, but we will not consider these two options in the present work.

The Aggregators are capable of joining several micro flex-offers into larger macro flex-offers, which are then placed on the electricity market. The energy market will answer with bids to buy and sell energy at given times. Aggregators receive and respond to the bids which allocate energy consumption periods to the macro flex-offers. After, they disaggregate macro flex-offer responses and send an answer to the Prosumers/Users which specify the periods of time to consume the required energy amount from the grid at a lower cost. It is the QoS Module that has the responsibility to find the best matching between the Prosumer's request for a service and the Aggregator that best covers its needs, in terms of response time, availability, and cost.

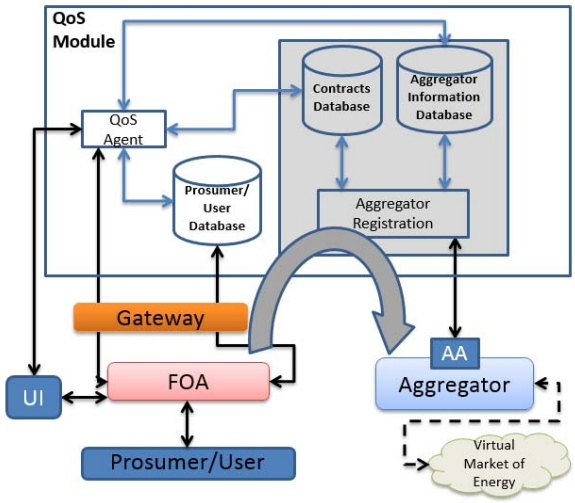

Figure 2 Proposed QoS Architecture

A. *Aggregator*

The Aggregator is responsible for the handling of flex-offers from the FOA, joining (aggregating) several micro flex-offers into a larger macro flex-offer, placing the macro flex-offer on the Virtual Market of Energy, disaggregating scheduled macro flex-offers, sending scheduled micro flex-offers to FOAs, controlling the execution of a micro scheduled flex-offer, determine if the execution of the flex-offer by the Device had been done according to the scheduled flex-offer. Each Aggregator can be specialized on different types of devices, by running the most adequate algorithms for the aggregation and disaggregation of flex-offers.

B. *Aggregator Agent (AA)*

Every Aggregator has an agent that provides information to the QoS Module. The Aggregator provides information to the QoS Module that has to do with the number of the users it is able to serve, possible cost of the provided service, time to respond to the Prosumer's request. It can also provide information regarding the type of services it can provide. It is common in the energy market to have a range of different type of Aggregators to cover the needs for home appliances (e.g., washing machines, heat pumps, etc.) and different ones to cover the needs of Electric Vehicles charging. This information is of great importance to the QoS Module in order to correctly and fast identify the most appropriate Aggregator to deliver the service to the Prosumer. The AA is indirectly connected to Flex-Offer Agent (FOA) via the QoS Module.

C. *Aggregator Registration*

The Aggregator Registration allows Aggregators, through the Aggregator Agent (AA) to submit: their id, service descriptions, cost functions, availability, and number of Prosumers/Users they can serve, to the QoS Module.

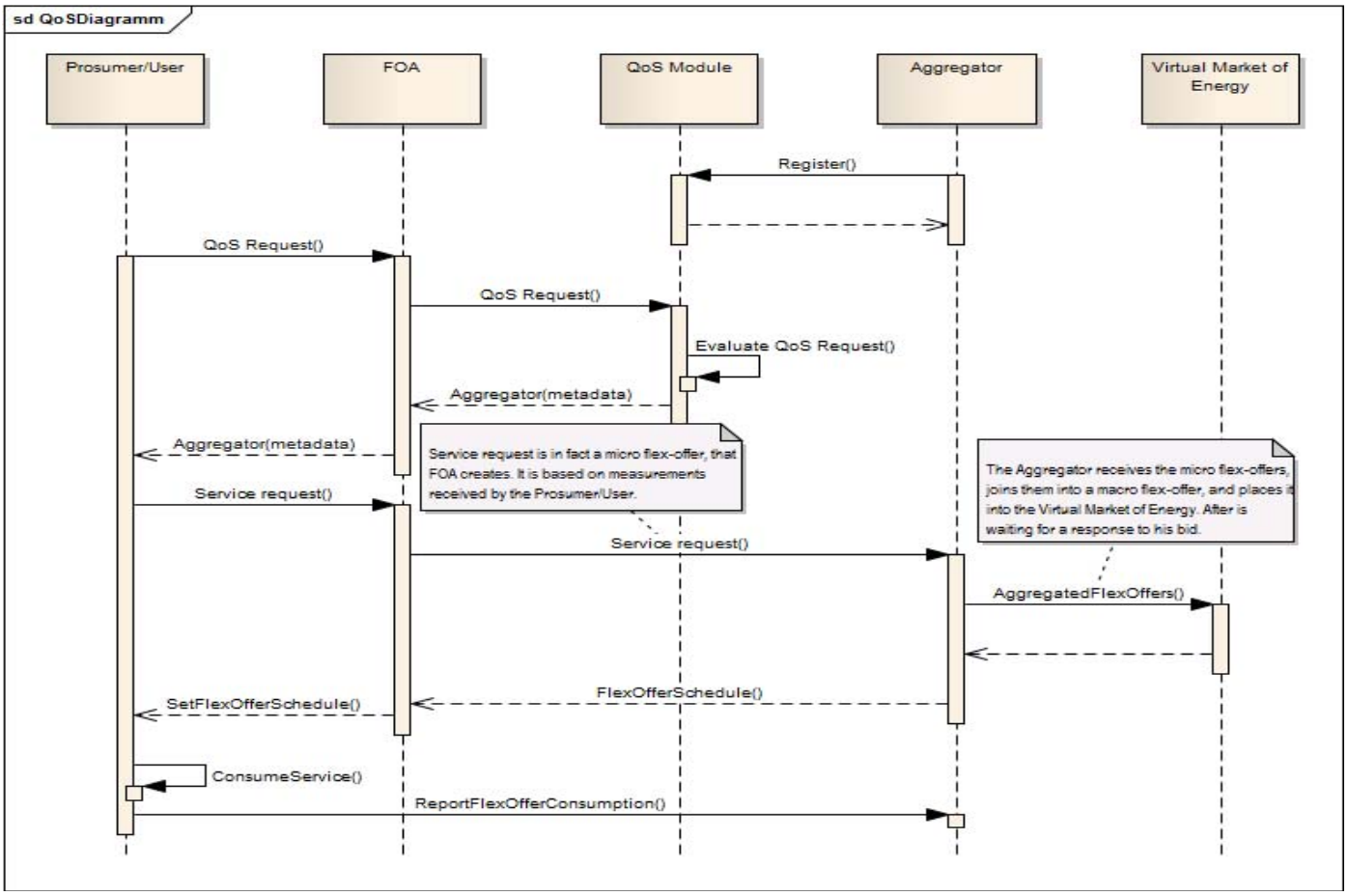

Figure 3 QoS delivery Sequence Diagram

D. Prosumer/User

A Prosumer (or User) owns devices and has an agreement with an Aggregator regarding utilizing the devices power consumption or production flexibility. Devices are the end equipment that consume or produce the energy belonging to a flex-offer, e.g., an EV, a heat pump or a washing machine. Devices can have the capability of being remotely controlled or might not have any computer interfacing capabilities. The Prosumer has to set up all relevant constraints/comfort requirements, which the flex-offer must fulfill. The Prosumer might be a household, factory, an office building, i.e. a legal entity that owns devices. A Prosumer uses a Flex-Offer Agent to generate flex-offers or it can configure these parameters through a user interface.

E. Flex-Offer Agent (FOA)

Every Prosumer/User has an agent that provides information to the QoS Module. FOA is a software module, which acts as an intermediate between Devices and Aggregators, being able to be executed on a variety of hardware platforms and easily configured to use different protocols. Based on constraints set up by the Prosumer and on power consumption measurements taken from devices it uses a specific algorithm to automatically generate micro flex-offers.

Other inputs like weather forecasts might also be used. The FOA can send the micro flex-offers to the Aggregator and receive the micro scheduled flex-offers from it. Another kind of information the Prosumer/User passes to the QoS is the type of service it needs (domestic appliances, heat pumps, or EVs). As in the case of the Aggregator Agent, this information is of great importance to the QoS Module in order to correctly and fast identify the most appropriate Aggregator to deliver the service to the Prosumer. The Flex-Offer Agent passes the request for a service to the QoS Module through the QoS Agent.

F. QoS Agent (QA)

QoS Agent (QA) is responsible for evaluating the Prosumer request, and identify an Aggregator that properly meets the client's needs. The QoS Agent receives the request from the Flex-Offer Agent (FOA) and evaluates the Prosumer/User request against each available Aggregator in order to identify the one that best fits the Prosumer/User needs. A Prosumer's request will probably contain a service type, cost constraint and the preferred comfort level. Once the time the mapping is succeeded the micro flex-offer is passed to the Aggregator to continue with the building of the macro flex-offers and the placement to the market of energy.

G. User Interface

The User Interface can take care of the interactions among the Prosumers/Users, the FOA, and QoS Agent through a web-based interface. It can be used to allow generation of flex-offers by a Prosumer/User or just to enforce attributes like a particular comfort level to the QoS Module.

H. Gateway

The Gateway can be seen as a device that converts between the protocols used internally on a Home Area Network and the internet. It is possible to have the capability of executing the Flex-Offer Agent.

I. Contracts and Aggreagator Information Databases

The Contracts Information Database is a databases to store SLAs, closed, scheduled, and served contracts. The Aggregator Information database is a database for keeping information regarding the Aggregators, Aggregator's information like type of services, availability, response time and cost models. Also the id of the Aggregator is stored on the Aggregators Information database. The id of the Aggregator is important in order the the Prosumer/User through the FOA, and the QoS Module to identify the correct one.

J. Prosumer/User Database

The Prosumer/User Database is a database that holds information regarding the Prosumers/Users. Information like: power consumption, type of Prosumer/User (flex-offer enabled or legacy device), if he was served or not.

K. QoS Module Interactions

The available Aggregators register themselves to the QoS Module and particular to the Aggregators Information Database, providing information like type of provided services, response time and cost models. The Prosumer asks for a service, which in our case is a need for energy consumption. This type of information is named micro flex-offer. It is then the responsibility of the QoS Module to perform all the needed steps in order to spot the Aggregator that best serves the needs of the Prosumer. Figure 3 (see p. 3) presents the interactions between the Prosumer, the QoS Module and the Aggregator:

1. Aggregators register themselves (with their id), and their services (type of services, response time, cost models, and number of Prosumers/Users each can serve) with the QoS Module.
2. A Prosumer/User initiates the sequence of steps, by sending to the QoS Module a QoS request (pointing out the requested service type, amount of needed energy, cost constraints, time flexibility).
3. The QoS Module identifies the Aggregator that best fits the needs of the Prosumer/User. The QoS Module creates a token that includes information like the id of the Aggregator, a session id, the service id, expiration date and time for the offer.
4. If the Prosumer accepts the offer, the QoS Module saves it in the Contract database. The Prosumer only needs the created token to request the service in the given time.
5. The Prosumer makes a service request to the Aggregator using the created token.
6. The Aggregator creates the macro flex-offer and places a bid to the Virtual Market of Energy. The market answers back with a schedule.
7. The Aggregator sends the Schedule to the Prosumer/User, through the Flex-Offer Agent.
8. The Prosumer consumes the service and reports back to the Aggregator the power consumption.

IV. Mining & Machine Learning Aspects to Smart Grid

In classical machine learning, the complexity and diversity of the field is controlled by the "Black-box" principle, where each machine learning method is expected to fit a simple mold. We will try to provide some insight to the "Black-box" in order to present its architecture and functionality. The Query Results Management (QRM) component/module will be responsible for managing the data that are extracted from the queries to the QoE system and for assembling the dataset that will be fed to the machine learning algorithm. In Figure 4, an illustration of where the QRM manager component is situated in relation to QoE's system and to the Machine Learning module is presented. The QRM module, a fundamental component of a more complete system, will be responsible for supporting the following functionalities:

1. Establishing a safe connection to the QoE databases;
2. Querying the databases, receiving the data; and
3. Saving the data in a file and in the proper format for the Machine Learning Management (MLM) module.

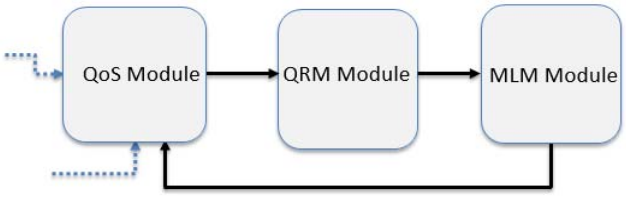

Figure 4 Proposed QoS Architecture

Even though many of the algorithms are different there are some common steps that should be followed while developing and applying a machine learning algorithm. These needed/common steps are the following:

1. **Data Collection:** Meaning the method for collecting the data. It varies from obtaining the data through an API, RSS feed, or even a device that collects data and sends them to you, etc.
2. **Data Preparation:** Making sure that the data are in a usable format. Some algorithms need features in a special format. Some can deal with features and variables as strings, and some others need them to be transformed into integers.

3. **Training the algorithm:** In this step, you feed the algorithm with "clean" data from the previous steps and obtain knowledge and insight from the data. In the case of unsupervised learning, there is no training step, since there is no target value.
4. **Testing the algorithm:** In this step, the evaluation of the algorithm takes place. In the case of supervised learning, you have known values for evaluating the algorithm (i.e. you have examples of data known as the ground truth that you can check against the performance of the algorithm). In unsupervised learning, there is a need to use other metrics like support and confidence to evaluate its success.
5. **Usage:** The actual implementation of the algorithm in practice that includes all the previous steps. There is also a need to continuously check if all the previous steps are working as expected. The QRM component will be responsible for the two first steps of the aforementioned procedure. Figure 5 presents the interactions of the QRM module with the Databases and the Machine Learning Module.

A. Query Results Management (QRM) Module

The QRM module will be responsible for the two first steps of the aforementioned procedure. Figure 5 presents the interactions of the QRM module with the Databases and the Machine Learning Module.

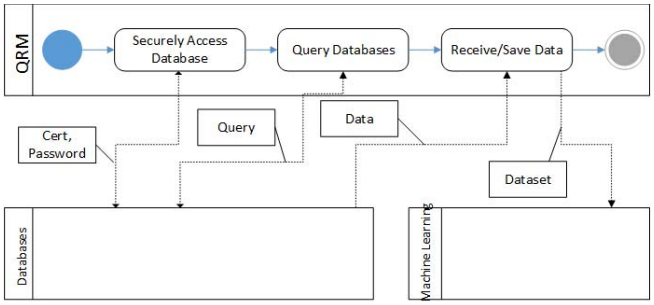

Figure 5 QRM component interactions

B. Machine Learning Management (MLM) Module

The MLM module will be responsible for the three last steps of the aforementioned procedure (steps 3 to 5). It will provide the needed functionality for the system to be in position to get the clean data, pass them to the machine learning algorithm and return useful conclusions. The MLM module should be in position to find interesting relationships in a large dataset. Quantifying interesting relationships is twofold. The first way is a frequent itemset, and the second is the one measuring interesting relationships in association rules.

One such approach is the Apriori [12] algorithm. Apriori uses the so-called Apriori principle to reduce the number of sets that are checked against the dataset. The Apriori principle denotes that if an item is infrequent, then supersets containing that specific item will be infrequent too. Apriori starts from single itemsets and creates larger sets by combining sets that meet the minimum support measure. Support is used to measure how often a set appears in the original dataset. Once frequent itemsets are found, someone may use them to generate association rules. The importance of an association rule is measured by the confidence. Confidence denotes the number that this rule applies to the frequent itemsets. The pseudocode of the Apriori algorithm is presented in Algorithm 1.

Algorithm1. The Apriori algorithm.

C_k: Candidate itemset of size k
L_k : frequent itemset of size k

(1) L_1 = {frequent items};
(2) **for** (k = 1; L_k != ∅; k++) **do begin**
(3) C_{k+1} = candidates generated from L_k;
(4) **for each** transaction t in database do
(5) increment the count of all candidates in
(6) C_{k+1} that are contained in t
(7) L_{k+1} = candidates in C_{k+1} with min_support
(8) **end**
(9) **return** $\cup_k L_k$;

Another approach is the FP-growth [14] algorithm. The FP-growth algorithm is another efficient way of finding frequent patterns in a dataset. Even though it follows the Apriori principle, it is much faster than the Apriori one, since it goes over the dataset only twice. The data is stored in an FP-tree structure. After you can find frequent itemsets by finding conditional bases for an item, and eventually building a conditional FP-tree. The aforementioned process is repeated, by conditioning on more items, until the conditional FP-tree has only one item. The pseudocode for the FP-growth algorithm is presented in Algorithm 2.

Algorithm2. The FP-growth algorithm.

Input: constructed FP-tree
Output: complete set of frequent patterns
Method: Call FP-growth (FP-tree, null).
Procedure FP-growth (Tree, α)
{
(1) **if** *Tree* contains a single path *P* then
(2) **for each** combination (denoted as β) of the nodes in the path *P* **do**
(3) generate pattern $\beta \cup \alpha$ with *support* = *minimum support of nodes* β
(4) **else for each** a_i in the header of *Tree* **do** {
(5) generate pattern $\beta = a_i \cup \alpha$ with *support* = a_i.*support;*
(6) construct β's conditional pattern base and then β's conditional FP-tree $Tree_\beta$
(7) **if** $Tree_\beta$!= ∅
(8) **then** call FP-growth ($Tree_\beta$, β)
}

V. CONCLUSIONS

In this paper we presented an outline for a Quality of Service architecture targeting the Smart Grid world. All the involving parts were in detail described and documented. QoS attributes like: type of service to be served, response time, availability, and cost where taken into consideration while sketching the proposed architecture. Future work will include definition of algorithms to be used for the QoS provisioning and implementation of the proposed architecture. Another equally important step is handling the different ways that a flex-offer can be generated and come up with an as common as possible approach. In this paper we considered the flex-offer to be created by the Flex-Offer Agent that is actually connected to the Prosumer/User. Other identified formal cases are the generation of the flex-offer on the Aggregator, by using power measurement data available on the cloud, and the flex-offer to be initiated by the Prosumer/User, through a User Interface provided by the Flex-Offer Agent. We also presented an initial supplementary architecture to mine the information stored in the databases and further process the data with the use of machine learning algorithms to extract useful information, like identifying common patterns amongst multiple users/prosumers. Common patterns for instance in electricity usage in terms of time and amount. In this way the market of energy will be in position to better regulate its production thus leading to a more stable and economically sustainable power grid.

ACKNOWLEDGMENT

Special thanks to former WP5 partners from the Arrowhead project [13], and to Prof. L.L. Ferreira from CISTER Research Unit for sharing their views and thoughts on defining a QoS architecture for the Smart Grid world.

REFERENCES

[1] E. Newcomer, G. Lomow, "Understanding SOA with Web Services," ISBN-10: 0321180860, ISBN-13: 9780321180865, Publisher: Addison-Wesley Professional, Copyright: 2005

[2] M. Rosen, B. Lublinsky, T.K. Smith, J.M. Balcer, Applied SOA: Service-Oriented Artchitecture and Design Strategies. John Wiley & Sons; Pub. Date: June 16, 2008 , Print ISBN: 978-0-470-22365-9; Web ISBN: 0-470223-65-0

[3] E. Thomas, SOA Design Patterns. Prentice Hall PTR, ISBN: 0136135161 (2009)

[4] L.L. Ferreira, L. Siksnys, P. Pedersen, P. Stluka, C. Chrysoulas, T. Guilly, M. Albano, A. Skou, C. Teixeira, T. Pedersen, "Arrowhead compliant virtual market of energy," in Emerging Technology and Factory Automation (ETFA), 2014 IEEE, Sept 2014, pp. 1–8 (2014).

[5] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, A. Roy, "A distributed resource management architecture that supports advance reservations and coallocation," in: Proc. Intl. Workshop Quality of Service 1999, UCL, 1–4 June, London, 1999, pp. 27–36. (1999)

[6] S. Ran, "A model for web services discovery with QoS," ACM SIGEcom Exchanges 4 (1) 1–10. (2003)

[7] M. Tian, A. Gramm, T. Naumowicz, H. Ritter, J. Schiller, "A concept for QoS integration in web services," in: Fourth Intl. Conf.Web Information Systems Engineering Workshops, WISEW'03, Roma, Italy, December 2003, pp. 149–155. (2003)

[8] F. Siqueira, V. Cahill, "Quartz: A QoS architecture for open systems," in: The 20th Intl. Conf. Distributed Computing Systems, ICDCS 2000, 10–13 April, Taipei, Taiwan, 2000, pp. 197–204 (2000)

[9] D.L. Tien, O. Villin, C. Bac, "Resource managers for QoS in CORBA," in: Second IEEE International Symp. Object-Oriented Real-Time Distributed Computing, 2–5 May, Saint-Malo, France, 1999, pp. 213–222. (1999)

[10] T. Cucinotta, A. Mancina, G. Anastasi, G. Lipari, L. Mangeruca, R. Checcozzo, F. Rusinà, "A real-time service-oriented architecture for industrial automation," IEEE Trans. Ind. Informat., vol. 5, no. 3, pp. 267-277 (2009)

[11] C. Cavanaugh, L.R. Welch, B. Shirazi, E. Huh, S. Anwar, "Quality of service negotiation for distributed, dynamic real-time systems," in: IPDPS Workshop on Bio-Inspired Solutions to Parallel Processing Problems, BioSP3, 15 April, Fort Lauderdale, FL, 2002, pp. 757–765 (2002)

[12] R. Agrawal, R. Srikant, "Fast algorithms for mining association rules," in: Proc. 20th Int. Conf. Very Large Data Bases, VLDB, 1994, pp. 487–499 (1994)

[13] The Arrowhead project: http://www.arrowhead.eu/ [accessed May 2016]

[14] H. Li, Y. Wang, D. Zhang, M. Zhang, and E. Chang, "PFP: Parallel FP-Growth for Query Recommendation," RecSys 2008, Proceedings of the 2008 ACM Conference on Recommender Systems, pp. 107-114. (2008)